presented at Autonomous Agents 2001
Montreal, Canada, May 28 – June 1, 2001

Specifying Agent Behavior as Concurrent Tasks

Scott A. DeLoach
Department of Electrical and Computer Engineering
Air Force Institute Of Technology
Wright-Patterson Air Force Base, Ohio 45433-7765
sdeloach@computer.org

ABSTRACT

Software agents are currently the subject of much research in many interrelated fields. Unfortunately, there has not been enough emphasis on defining the techniques required to build practical agent systems. While many agent researchers refer to tasks, few really define what they mean. Tasks not only define the internal processing an agent must perform, but also how interactions with other agents relate to internal processes.

1. Introduction

Many agent researchers refer to tasks performed by roles within a multiagent system. However, few really define the essence of what they mean by tasks. We believe that the definition of tasks is critical to define completely the behavior of multiagent system. Tasks not only define the types of internal processing an agent must do, but also how interactions with other agents relate to those internal processes. Some researchers have focused on coordination and some on internal agent reasoning, few have combined the two.

In general, our research has focused on developing the methodology, techniques, and tools for building practical agent systems [1]. To this end, we have developed the Multiagent Systems Engineering methodology [6] that defines multiagent systems in terms of agent classes and their organization. We define their organization in terms of which agents can communicate using conversations. There are two basic phases in MaSE: analysis and design. The first phase, Analysis, includes three steps: capturing goals, applying use cases, and refining roles. In the Design phase, we transform the analysis models into constructs useful for actually implementing the multiagent system via four steps: creating agent classes, constructing conversations, assembling agent classes, and system design. In this paper, we present concurrent tasks, which we use in the analysis phase to define the internal processing of communications of roles. A more complete definition of concurrent tasks is found in [2].

2. Concurrent Tasks

We define agent behavior to by a set of *n* concurrent tasks. Each *task* specifies a single thread of control that defines the behavior of an agent and integrates inter-agent as well as intra-agent interactions. We typically think of concurrent tasks as defining how a role decides what actions to take, not necessarily what the agent does. This is an important distinction when talking about agents since hard-coding specific behavior may not be the ideal case. Often agents incorporate the concept of plans and planning to determine what to do. In these cases, we would develop a concurrent task for determining how the planning and plan implementation occurs, but not for describing the individual plans themselves. Concurrent tasks are specified graphically using a finite state automaton as shown in Figure 1. Tasks that start with *null* transition from the start state are assumed to start execution upon startup of the agent and continue until the agent terminates or an end state is reached. Tasks that have a receive event on the initial transition are assumed to be reactive and start upon the receipt of a particular message.

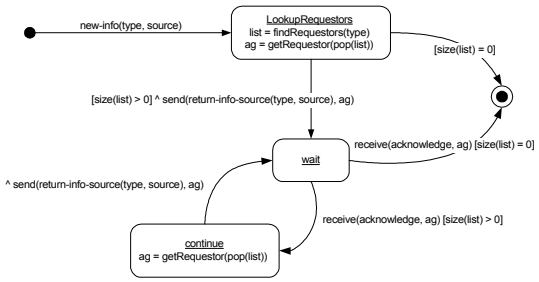

Figure 1. Inform Requestors of New Source Task

Activities are used inside states to specify functions carried out by the agent. While tasks execute concurrently and carry out high-level behavior, they are coordinated using internal events. Internal events are passed from one task to another and are specified on the transitions between states. To communicate with other agents, external messages can be sent and received. These are specified as *send* and *receive* events, which send and retrieve messages from the message-handling component of the agent, which is assumed to exist. Besides communication with other agents, tasks can interact with the environment via reading percepts or performing operations that affect the environment. This interaction is typically captured by activities executed within the task states. By including reasoning within tasks, agents are not "hardwired" or purely reflexive. They can plan, search, or use knowledge-based reasoning to decide on appropriate actions

Concurrent tasks have two components: states and transitions. These states and transitions are similar to other finite automata states and transitions. Transition syntax is shown below.

```
trigger [guard] ^ transmission(s)
```

A *trigger* is an event received from another agent or concurrent task, the *guard* is a Boolean condition, and the *transmission* represents the sending of an event to another concurrent task or a message to another agent. Two special events are used to indicate that a message is actually sent/received to/from another agent: *send* and *receive*. The send event is used to send a message to an agent and has the following syntax.

```
send(message, agent)
```

For example, in the transition from `LookupRequestors` to `wait`, `send(return-info-source(type, source),ag)`, denotes a transmission. In this case, if the condition `size(list)>0` is true, this is a message to `agent` to `return-info-source(type, source)`. In this case, `return-info-source` is the performative while `(type, source)` defines the message content. The syntax of the receive event is shown below.

```
receive(message, agent)
```

In this case, a receive event is only valid as a trigger and follows the same syntax rules as the send event.

States may contain activities (represented as functions), which can be used to represent internal reasoning, reading a percept from sensors, or performing actions via effectors. Multiple activities may be included in a single state and are performed in sequence. Once in a state, the task remains in that state until activity processing is complete and a transition out of the state becomes enabled.

3. Task Types

As stated initially, the goal of concurrent tasks is to define the behavior of agents, tying the internal reasoning processes of the agent to its interaction with other internal processes as well as externally with other agents. Based on the semantics presented in the previous section, we can categorize these tasks by their *life span* and their *responsiveness*.

There are two types of task life spans: persistent or transient. A *persistent* task is a task that has a *null* transition from the start state to the first state – it does not have an event that initiates its execution. We assume that persistent tasks start when the agent is initiated and continue until the agent or the task terminates. On the other hand, a *transient* task has a specific trigger on the transition from the start state. A transient task is not executed when the agent starts, but waits until its trigger is received by the agent. With *transient* tasks, it is possible to have multiple, concurrently executing tasks of the same type.

As far as responsiveness, a task may be reactive, proactive, or heterogeneous. A *reactive task* either has an idle state where it waits for a triggering event before actually starting any processing, or is a transient task that starts executing in response to event. *Proactive tasks* do not have idle states and are not transient. They are continually generating requests for other agents or tasks. A *heterogeneous task*, as the name suggests, is a combination of reactive and proactive tasks. A heterogeneous task may have idle states, but does not start in an idle state. It generates at least one request for another agent or task before entering an idle state.

Based on these task definitions, we can categorize agent whose behavior is defined by tasks as either proactive or reactive. A *proactive agent* is an agent with at least one proactive or heterogeneous task while a *reactive agent* is an agent whose tasks are all reactive.

4. Related Work

Much of our work on Concurrent Task Models stems from work originally done by Harel on Statecharts [4], which is the basis for state diagrams in many of the current object oriented modeling languages. Statecharts are a very large, complex language supporting concurrency, conditional transitions, and event input and output. The basic difference between Concurrent Tasks and Statecharts is the ability to define parameterized events and activities inside states in Concurrent Tasks. Concurrent tasks are also similar to Dooley graphs for agent coordination [5]. Another approach to modeling behavior and coordination in multiagent systems is the use of Petri nets, such as the Ferber's BRIC formalism [3]. While Petri nets make the parallelism between tasks and agents explicit, to the less trained practioner they can be more difficult to use and understand.

5. Conclusions

Concurrent Tasks are the central behavioral model used in the analysis phase of the Multiagent Systems Engineering methodology. By analyzing the system as a set of roles and tasks, a system designer is lead naturally to the definition of autonomous, pro-active agents that coordinate their actions to solve the overall system goals. Future work on MaSE and Concurrent Tasks include the automatic transformation of the behavior modeled by concurrent tasks into concrete designs, and eventually source code. So far, Concurrent Tasks have been used, as part of MaSE, to analyze and design a number medium sized multiagent systems ranging from information systems, distributed mixed-initiative planners, biologically-based immune systems, to control systems for autonomous uninhabited air vehicles.

6. Acknowledgements

The Air Force Office of Scientific Research sponsored this research. The views expressed in this article are those of the authors and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the US Government.

7. References

[1] DeLoach, S. A., and Wood, M. Developing Multiagent Systems with agentTool. The Seventh International Workshop on Agent Theories, Architectures, and Languages (ATAL-2000). Boston, MA, July 7-9, 2000.

[2] DeLoach, S. A., Specifying Agent Behavior as Concurrent Tasks: Defining the Behavior of Social Agents. Technical Report, Air Force Institute of Technology, AFIT/EN-TR-00-03, July 2000.

[3] Ferber, J. Multi-Agent Systems: an Introduction to Distributed Artificial Intelligence. Addison Wesley Longman, 1999.

[4] Harel, E., and Politi, M. Modeling Reactive System with Statecharts: the Statemate Approach. McGraw-Hill, New York, New York. 1998.

[5] Parunak, H. V. D. Visualizing agent conversations: Using Enhanced Dooley graphs for agent design and analysis. In Proceedings of the 2nd International Conference on Multiagent Systems, pages 275-282. AAAI Press, 1996.

[6] Wood, M. F., and DeLoach, S. A. An Overview of the Multiagent Systems Engineering Methodology. The First International Workshop on Agent-Oriented Software Engineering (AOSE-2000), Limerick Ireland, June 2000.